caCIS Secure Email Design

caBIG® Clinical Information System (caCIS) Secure Email Design Document v1.0.1

Contributors

Authors	Kunal Modi
Reviewers	Santosh Joshi Harsh Marwaha Lloyd McKenzie
Editor	Lauren Anthone

Document Change History

VersionNumber	Implemented By	Revision Date	Description of Change
0.1	Kunal Modi	08/17/2011	Initial Draft
0.2	Kunal Modi	08/25/2011	Incorporated feedback from internal review
1.0	Kunal Modi	09/30/2011	Baselined
1.0.1	Lauren Anthone	10/18/2011	Converted document to wiki and edited content

Table of Contents

1. Introduction
2. Overview of S/MIME
3. Detailed Design for Transmitting Secure Emails
    - 3.1. Creating an Email with the Attachment
    - 3.2. Storing Recipient's Public Certificates
    - 3.3. Encrypting Email Using the Recipient's Public Certificate
    - 3.4. Storing Sender's Public-Private Key Pair
    - 3.5. Signing Using Sender's Private Key
    - 3.6. Transmitting the Message via Email

1. Introduction

The caCIS platform enables the source clinical system to request delivery of a particular formatted publication of patient's data to be received by the recipient target system. There are multiple mechanisms in which this transmission can occur, such as Email, File Transfer and using IHE's XDS and NAV profiles.

All of these interfaces must be secured in order to restrict access to patient data to authorized users only. This document provides details about securing the Secure Email interface transmission mechanism. XDS/NAV and Secure File Transfer are covered in separate design documents.

S/MIME as recommended by NHIN Direct has been chosen as the mechanism for transferring documents via secure email. This document focuses on the design for achieving such secure email transmission.

2. Overview of S/MIME

S/MIME (Secure Multipurpose Internet Mail Extensions) was originally proposed by RSA Data Security, Inc. in 1995, which then led an industry consortium, including most of the major e-mail software and Internet browser vendors such as Microsoft, Netscape and Lotus. Development work

is now being coordinated by the IETF S/MIME Working Group.

S/MIME can be used by traditional mail user agents (MUAs) to add cryptographic security services to mail that is sent, and to interpret cryptographic security services in mail that is received. However, S/MIME is not restricted to mail; it can be used with any transport mechanism that transports MIME data, such as HTTP. As such, S/MIME takes advantage of the object-based features of MIME and enables secure messages to be exchanged in mixed-transport systems.

3. Detailed Design for Transmitting Secure Emails

Transmitting an email securely involves the following sub tasks:

1. Create an email that contains the To, From and the Document to be transferred as an attachment
2. Encrypt the email message with the recipient's public key so that only intended recipients can decrypt the message using their corresponding private key
3. Sign the email message with sender's private key to ensure the following:
    a. The message verifiably originated from the sender
    b. The content of the message was not tampered with during transmission
4. Send the encrypted and signed email message containing the document as an attachment

The following sub sections detail these tasks and how they will be performed.

3.1. Creating an Email with the Attachment

The java component invoked by the document router receives the following parameters:

- Recipient's email address
- Incoming caCIS Message
- Document to be transmitted

This component can use the JavaMail APIs to perform the following:

1. Read the sender's address from the configuration file.

    

    A sender's address must be configured for each integration platform installation. This sender's address is common for all document transfers and is shared by the NAV Notification sending mechanism as well. This can be configured in a database or a flat file.

2. Create a new email message using the passed recipient's email address in the TO section.
3. Set the subject to indicate the type of document that is being transferred (from the recipient's desired document type as specified in the routing instructions in the caCIS Request).
4. Add message body details that can contain details about the document, such as the document type; the document metadata such as the study, site and patient information; etc. These are obtained from the passed caCIS Request data.
5. Attach the document, which is passed as the parameter, to this message.

This creates a MIME message with the following components

- Sender's From address
- Recipient's To address
- Subject line
- Message body
- Document attachment

The email can then be encrypted and transmitted as a secure email.

3.2. Storing Recipient's Public Certificates

One of the major tasks in sending secure emails is to be able to obtain a recipient's public certificate in order to use it for encrypting the message. This ensures that only the recipient can decrypt the message using a unique private key. There are various standards that can support storage and retrieval of such public certificates, one of which is the RFC 2538- Storing Certificates in the Domain Name System (DNS) (http://tools.ietf.org/html/rfc2538). N-HIN Direct recommends and uses this standard. However this requires setting up, administering and managing a DNS Service, all of which requires a degree of effort.

To facilitate something similar within the caCIS Integration Solution, a trust store stores the recipient's public certificate along with the recipient's email address as an alias to the public certificate.

3.3. Encrypting Email Using the Recipient's Public Certificate

JavaMail API by default does not provide any encryption support for S/MIME. However, it does allow several third party APIs which can work in conjunction with JavaMail to add the S/MIME capabilities. JavaMail-Crypto is one of such API. It is a simple, easy-to-use API that provides a unified way to access S/MIME encryption functionality with JavaMail. It uses bountycastle jars underneath to provide the actual encryption and signing capabilities.

Once the MIME message has been formulated, it must be encrypted using the recipient's public key. The following steps will be performed:

1. Retrieve the MIME Message which is created in Creating an Email with the Attachment. This message contains the document to be transmitted
2. Using the recipient's email address from the TO section of the mail, retrieve the public certificate for the recipient from the trust store described in Storing Recipient's Public Certificates.
3. Use the JavaMail-Crypto APIs to encrypt the MIME Message using the public key retrieved in the step above

As a result, the MIME message is encrypted using recipient's public key.

3.4. Storing Sender's Public-Private Key Pair

In order to sign the email message, the sender must have a public private key pair. This can be generated using java keytool or can be obtained from third party vendors. Once the key pair is obtained, it is stored in a key store with the sender's email address as the alias. The sender's private key can then be retrieved by passing the sender's email address as the alias.

3.5. Signing Using Sender's Private Key

Once the email has been encrypted, it must be signed using the sender's private key. The following steps are performed to sign the encrypted MIME Message:

1. Retrieve the encrypted MIME Message which is created in Encrypting Email Using the Recipient's Public Certificate.
2. Using the sender's email address from the FROM section of the mail, retrieve the private key for the sender from the key store described above in Storing Recipient's Public Certificates.
3. Use the JavaMail-Crypto APIs to sign the encrypted MIME Message using the private key retrieved in the step above.

As a result, the message is signed using sender's private key.

3.6. Transmitting the Message via Email

An instance of SMTP server must be configured for caCIS Integration Platform. Once the email has been encrypted and signed, it is ready for transmission. The JavaMail APIs are used to send the message using the available SMTP Server.